YVR18-417: Struck entropy!
Finding true randomness from sensor data

Sumit Garg and Daniel Thompson
Agenda

- Platform: Developerbox
- OP-TEE port
- OP-TEE use-case: RNG feasible?
- RNG mechanism
- Is this RNG truly random?
- Optimize RNG collection
- RNG use-cases
Platform: Developerbox

Based on Socionext SynQuacer SC2A11 multi-core chip with 24 cores of ARM® Cortex-A53.

Use-cases:
- ARM® based software development environment.
- IoT gateway
- Edge computing
- Low power consumption server.
OP-TEE port for Developerbox

Note: Here numbering represents the boot sequence
OP-TEE use-case: RNG?

RNG – Random Number Generator

Developerbox lacks a hardware based TRNG.

Kernel provides a software implementation using randomness from inter-interrupt timings with following shortcomings:

- Lacks sufficient entropy at critical points (especially at boot)
- Not trusted (eg. by OP-TEE)
- Quite slow (especially when there are few interrupts)
OP-TEE use-case: RNG feasible

Developerbox provides 7 on-chip thermal sensors, accessible from secure world only, sensing temperature from various group of core clusters.

Do these thermal sensors contain sufficient noise to develop a TRNG?
Sensor: Randomness sources

Randomness (measurement error + ADC conversion error) resides in Least Significant Bits (LSBs) of sensor output depending on precision of measurement and ADC conversion.
# Thermal sensor: raw data

<table>
<thead>
<tr>
<th>Sample no. (freq: 2ms)</th>
<th>Raw binary data</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>1 0 1 0 0 0 0 0 0</td>
</tr>
<tr>
<td>S2</td>
<td>1 0 0 1 1 1 1 1 1</td>
</tr>
<tr>
<td>S3</td>
<td>1 0 1 0 0 0 1 0</td>
</tr>
<tr>
<td>S4</td>
<td>1 0 1 0 0 0 1 1</td>
</tr>
<tr>
<td>S5</td>
<td>1 0 1 0 0 0 0 0 1</td>
</tr>
<tr>
<td>S6</td>
<td>1 0 1 0 0 0 1 0</td>
</tr>
<tr>
<td>S7</td>
<td>1 0 1 0 0 0 0 0 1</td>
</tr>
<tr>
<td>S8</td>
<td>1 0 1 0 0 0 1 1</td>
</tr>
<tr>
<td>S9</td>
<td>1 0 1 0 0 0 1 0</td>
</tr>
<tr>
<td>S10</td>
<td>1 0 1 0 0 0 1 1</td>
</tr>
</tbody>
</table>
Collect randomness from least significant bits

RNG byte stream

Sampling sensor data every 2ms
Is this RNG truly random?

Answer to this really depends on how much paranoia one has. In our case we used following procedure to measure randomness:

- Collected approx. 2.1GB raw data from thermal sensors.
- Used “rngtest” (implements FIPS 140-2 RNG fitness tests).
RNG algo 1: LSB only

FIPS test (rngtest) results show 2.13% success ratio.

Collect randomness from least significant bits

Sampling sensor data every 2ms
RNG algo 2: LSB + xor of bit1

FIPS test (rngtest) results show 0.07% success ratio.

Collect randomness from least significant bits

Sampling sensor data every 2ms
RNG algo 3: LSB + xor of bit1 + CRC

Statistical fine tuning using CRC32 algo per 4 byte payload (32:32 bit whitening).

Statistical fine tuning using CRC32 algo per 5 byte payload (40:32 bit whitening).

Collect randomness from least significant bits

Sampling sensor data every 2ms
RNG testing results

- FIPS test (rngtest) results show **99.91%** success ratio.
- Entropy collection rate: **~500 bytes/sec**
- Tried using the dieharder suite to discriminate between the 32:32 bit and the 40:32 bit CRC whitening but results are ambiguous.
Comparison with random.org data

- FIPS test (rngtest) results show similar **99.92%** success ratio.
Aside: My Intel machine’s RNG

- Uses “rdrand” hardware instruction to get entropy.
- FIPS test (rngtest) results show similar **99.91%** success ratio.
- Entropy collection rate far higher: ~**1 Mbytes/sec**
RNG implementation

RNG pseudo TA built as part of OP-TEE OS. Initialized during OP-TEE init. Normal world client interface remains same as with Dynamic TA.
Issue with RNG collection

● Sensor values refresh every 2ms
  ○ Theoretic maximum RNG rate of ~500 bytes/sec is relatively low
  ○ Max rate is only achieved if we read (poll) the sensors frequently

● Earlier implementation relied on continuous busy looping in pseudo TA until requested RNG data is generated (time: 2ms * no. of bytes)
  ○ Busy looping for multi-milliseconds is wasteful

Busy looping? Yuck!
Optimize RNG collection

Configured OP-TEE pseudo TA with secure timer interrupts (freq: every 2ms) and entropy pool (size: 4k).

Functionality:

- At each interrupt, pseudo TA formulates byte from LSBs of thermal sensor output.
- Entropy pool is used to collect these bytes. Once pool is full, interrupts are disabled and enabled again with every entropy request.
RNG: SMC interface

**Normal world driver**
- Allocate non-sec SHM buffer (max. 4k)
- Issue request to get N bytes of rng data
- Does some useful work for \((N-E)\times2\text{ms}\) and then request again to get \((N-E)\) bytes
- Issue request to close session with RNG TA

**Secure world RNG service (TA)**
- open session
- ret: session
- invoke cmd
- ret: rng data
- invoke cmd
- ret: rng data
- close session
- ret: success

- Entropy pool (size: 4k)
- Creates session with RNG TA
- Available entropy \(E\) bytes in pool less than requested. Return \(E\) bytes and enable timer FIQ to collect entropy in backend.
- Returns \((N-E)\) bytes of entropy generated in the pool.
- Timer FIQ keeps on running until pool is full.
RNG use-case - Kernel: /dev/random

- **Normal World**
  - /dev/hwrng
  - TEE Client
  - TEE client API
  - Linux OS
  - UEFI (edk2)

- **Secure World**
  - Trusted Application [TA]
  - TEE Internal API
  - OP-TEE OS
  - pseudo TA
  - OPTEE Dispatcher
  - Trusted Firmware - A
  - SCP Firmware (CM-3)

- **Additional Components**
  - rngd daemon
  - SMC interface
  - Timer, FIQ
  - Thermal Sensor
  - Timer, FIQ
  - Thermal Sensor
“rngtest” results for /dev/random

sumit@oak:~/latest$
sumit@oak:~/latest$ sudo rngtest -c 1000 < /dev/random
rngtest 2-unofficial-mt.14
Copyright (c) 2004 by Henrique de Moraes Holschuh
This is free software; see the source for copying conditions. There is NO warranty;
rngtest: starting FIPS tests...
rngtest: bits received from input: 20000032
rngtest: FIPS 140-2 successes: 1000
rngtest: FIPS 140-2 failures: 0
rngtest: FIPS 140-2(2001-10-10) Monobit: 0
rngtest: FIPS 140-2(2001-10-10) Poker: 0
rngtest: FIPS 140-2(2001-10-10) Runs: 0
rngtest: FIPS 140-2(2001-10-10) Long run: 0
rngtest: FIPS 140-2(2001-10-10) Continuous run: 0
rngtest: input channel speed: (min=1.535; avg=2.338; max=5709.222)Kibits/s
rngtest: FIPS tests speed: (min=11.207; avg=11.391; max=11.560)Mbits/s
rngtest: Program run time: 8356993481 microseconds
sumit@oak:~/latest$
RNG use-case - Boot time: UEFI

Normal World

Client Application (CA)
- TEE client API
- OP-TEE Client
- TEE Driver
- Linux OS
- KASLR
- OP-TEE Library
- UEFI (edk2)
- Random seed

Secure World

Trusted Application [TA]
- TEE Internal API
- OP-TEE OS
- pseudo TA
- OPTEE Dispatcher
- Trusted Firmware - A
- SCP Firmware (CM-3)

FIRMWARE

USER

SMC interface

Timer

FIQ

Thermal Sensor
Next steps...

- Improvements to the whitening algorithm.
- Work towards creating a generic RNG interface to secure world handling:
  - Fast vs. slow entropy sources.
  - Implements entropy pool vs. must be called every N ms.
- Upstream RNG driver in edk2 (UEFI) and Linux.
Thank You

#YVR18
YVR18 keynotes and videos on: connect.linaro.org
For further information: www.linaro.org
Contact: support@linaro.org